複雑系科学演習　レポート

公立はこだて未来大学　システム情報科学部　B3　日置竜輔

2021 年 11 月 6 日

目次

1　複雑系科学演習：レポート課題 1

1.1　課題 1

ロジスティック写像の時系列変化を計算するプログラムを作成し、$r = 1.50$, $r = 2.60$, $r = 3.20$, $r = 3.50$, $r = 3.86$, $r = 3.90$ のとき、$x_0 = 0.7$ として個体数変動の時系列グラフを表示せよ。

1.2　時系列グラフの描画

時系列グラフは以下のようになる。

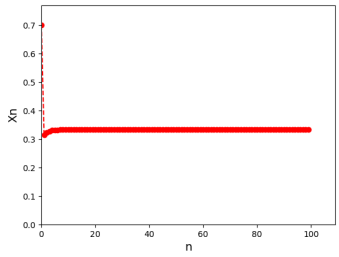


図 1 r=1.50 のときの時系列グラフ

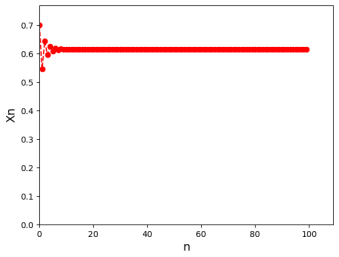


図 2 r=2.60 のときの時系列グラフ

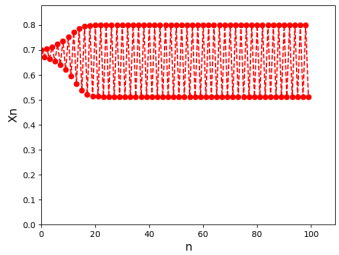


図 3 r=3.20 のときの時系列グラフ

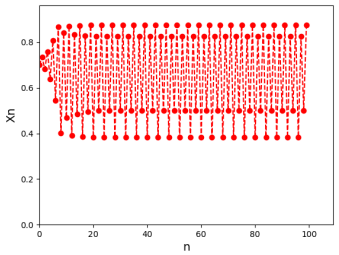


図 4 r=3.50 のときの時系列グラフ

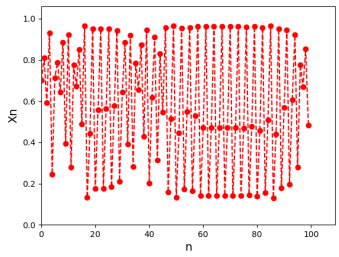


図 5 r=3.86 のときの時系列グラフ

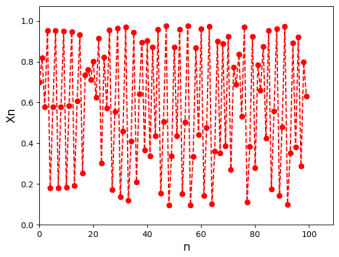


図 6 r=3.90 のときの時系列グラフ

1.3 時系列グラフのソースコード

課題１とに関するコードは以下のようになる。

Listing 1　Python による時系列グラフの描画

```
from matplotlib import pyplot as plt
from copy import copy
from collections import deque

class LogisticMap:

    def __init__(self, start: float, constant: float, number: int, problem_number: int)
        :初
        項
        ", 定数（の値）a, 長さ, 問題番号"
        self.num = [i for i in range(number)]
        tmp = start
        self.n = deque([(tmp := constant * tmp * (1 - tmp)) for _ in self.num])
        self.n.appendleft(start)
        self.n.pop()
        self.n = list(self.n)
        self.problem_number = problem_number

        x, y = copy(self.n), copy(self.n)
        x.pop()
        y.pop(0)
        self.x = x
        self.y = y

    # 時系列グラフ
    def time_series_graph(self):
        fig = plt.figure()
        plt.plot(self.num, self.n, marker="o", color="red", linestyle="--")
        plt.xlabel("n",fontsize=14)
        plt.ylabel("Xn",fontsize=14)
        plt.xlim(0.0, max(self.num) * 1.1)
        plt.ylim(0.0, max(self.n) * 1.1)
        plt.savefig(f"/Users/ryusuke/Documents/git/FutureUniversityHakodate/
            ComplexScienceExercise/result/時系列グラフ
            week01__{self.problem_number}.png")
        plt.close(fig)

    # リターンマップ
    def return_map(self):
        fig = plt.figure()
        plt.plot(self.x, self.y, marker="o", color="blue", linestyle="None")
```

```
38          plt.plot([i*0.01 for i in range(130)], [i*0.01 for i in range(130)], marker='
                o', color='orange', linestyle='None')
39          plt.xlabel("Xn",fontsize=14)
40          plt.ylabel("Xn+1",fontsize=14)
41          plt.xlim(0.0, max(self.x) * 1.1)
42          plt.ylim(0.0, max(self.y) * 1.1)
43          plt.savefig(f"/Users/ryusuke/Documents/git/FutureUniversityHakodate/
                ComplexScienceExercise/result/リターンマップ
                week01__{self.problem_number}.png")
44          plt.close(fig)
45
46  #  (1)
47  l1 = LogisticMap(0.7, 1.5, 100, 1)
48  l1.time_series_graph()
49
50  #  () 2
51  l1 = LogisticMap(0.7, 2.6, 100, 2)
52  l1.time_series_graph()
53
54  #  () 3
55  l1 = LogisticMap(0.7, 3.2, 100, 3)
56  l1.time_series_graph()
57
58  #  () 4
59  l1 = LogisticMap(0.7, 3.5, 100, 4)
60  l1.time_series_graph()
61
62  #  () 5
63  l1 = LogisticMap(0.7, 3.86, 100, 5)
64  l1.time_series_graph()
65
66  #  () 6
67  l1 = LogisticMap(0.7, 3.9, 100, 6)
68  l1.time_series_graph()
```

1.4　課題 2

ロジスティック写像のリターンマップを描くためのプログラムを作成し、$r = 1.50$, $r = 2.60$, $r = 3.20$, $r = 3.50$, $r = 3.86$, $r = 3.90$ のとき、$x_0 = 0.7$ として個体数変動のリターンマッ プを表示せよ。
グラフには、$x_{n+1} = r(1 - x_n)x_n$と $x_{n+1} = x_n$ のグラフも表示すること。

1.5　リターンマップの描画

リターンマップは以下のようになる。

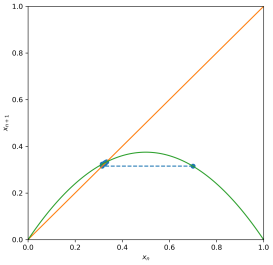


図 7 r=1.50 のときのリターンマップ

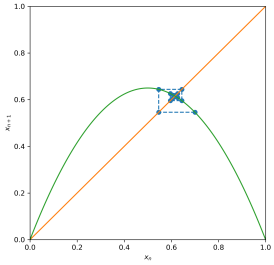


図 8 r=2.60 のときのリターンマップ

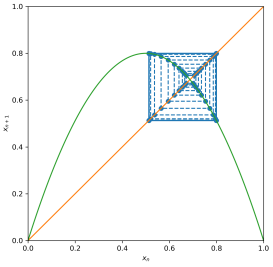


図 9 r=3.20 のときのリターンマップ

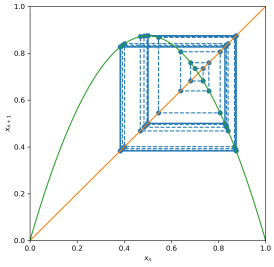


図 10 r=3.50 のときのリターンマップ

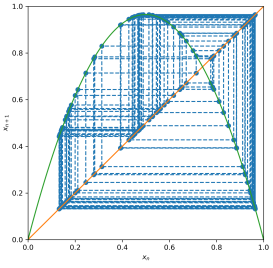


図 11 r=3.86 のときのリターンマップ

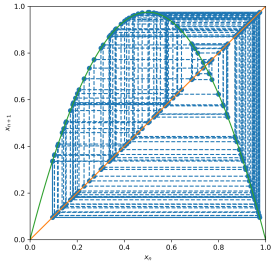


図 12 r=3.90 のときのリターンマップ

1.6 リターンマップのソースコード

課題２とに関するコードは以下のようになる。

Listing 2　Python によるリターンマップの描画

```
from matplotlib import pyplot as plt
import numpy as np

class LogisticGraph():
    def __init__(self, r: float, s: str) -> None:
        self.x = 0.7
        self.r = r
        self.s = s
        self.xn = np.linspace(0, 1, 1000)
        self.fig = plt.figure(figsize=(12, 6))
        self.filepath = '/Users/ryusuke/Documents/git/FutureUniversityHakodate/
            ComplexScienceExercise/result/リターンマップ
            week01_'

    def logistic(self) -> list:ロジスティック回帰の計算
        ''''''
        calc_x = self.x
        x_array = [calc_x]
        for _ in range(1, 102):
            calc_x = self.r * (1 - calc_x) * calc_x
            x_array.append(calc_x)
        return x_array

    def plot_return_map_only(self):リターンマップの描画
        ''''''
        self.fig = plt.figure(figsize=(6, 6))
        xn_array = []
        for i in self.xn:
            xn_array.append(self.r * (1 - i) * i)
        n = self.logistic()
        spiper_plot_x, spiper_plot_y = [], []
        for i in range(1, len(n)):
            spiper_plot_x.append(n[i - 1])
            spiper_plot_x.append(n[i])
            spiper_plot_y.append(n[i])
            spiper_plot_y.append(n[i])

        plt.plot(spiper_plot_x, spiper_plot_y, marker='o', linestyle='dashed')
        plt.plot(self.xn, self.xn)
        plt.plot(self.xn, xn_array)
        plt.xlim(0, 1)
```

```
        plt.ylim(0, 1)
        plt.xlabel("$x_n$")
        plt.ylabel("$x_{n+1}$")
        plt.savefig(self.filepath + '_' + self.s, dpi=300)

r = [1.50, 2.60, 3.20, 3.50, 3.86, 3.90]

for index, j in enumerate(r):
    LogisticGraph(j, str(index + 1)).plot_return_map_only()
```

1.7　レポート課題１に関するソースコードの補足

本レポートは全て Python で描画を行った。C 言語よりもグラフや数値計算に強いため、Python をとても強く推奨する。class を定義することで、時系列グラフやリターンマップに共通している部分を使い回すことが可能であるため、とても楽にコードを書くことが出来た。
漸化式の計算に関しては、再帰的に計算をしていくことで、計算量が O($(\frac{1+\sqrt{5}}{2})^N$) から O(N) で抑えることが出来たので、とても高速にコードを回すことが可能となった。

1.8　レポート課題１に関する考察

r の値が増加すればするほど、時系列グラフは発散していく傾向にあり、振動する結果が得られた。
カオスの性質から r の値を少し変更しただけでも描画されたグラフに大きな違いが見られたことは大きな発見であった。
リターンマップの性質としては、r の値を大きくしていけばいくほど、描画される点の数が多くなることがわかった。また、この点は $X_{n+1} = r(1 - X_n)X_n$ の二次関数に近似されていくことが読み取れた。

2　複雑系科学演習：レポート課題 2

2.1　課題 1

ロジスティック写像で $r = 1.50, r = 2.60, r = 3.20, r = 3.50, r = 3.86, r = 3.90$ として、初期値 x_0 を 0 から 1 まで 0.001 きざみで変化させたときの、x_{200} の値がどうなっているかグラフ化せよ。
また、x_n が $150 < n < 200$ の場合もグラフ化せよ。出力形式は授業資料を参照すること。

2.2　ロジスティック写像の描画

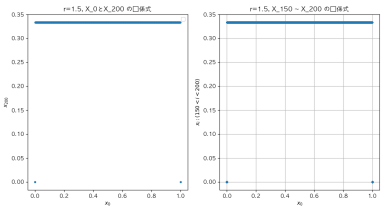


図 13 r=1.50 のときのロジスティック写像

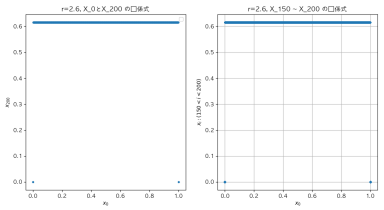


図 14 r=2.60 のときのロジスティック写像

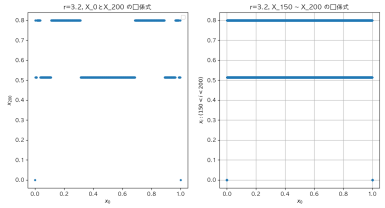


図 15 r=3.20 のときのロジスティック写像

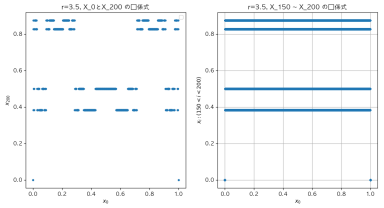


図 16 r=3.50 のときのロジスティック写像

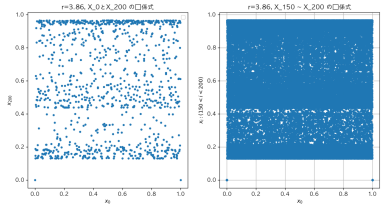


図 17 r=3.86 のときのロジスティック写像

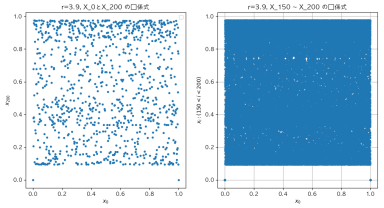


図 18 r=3.90 のときのロジスティック写像

2.3 課題１のソースコード

Listing 3　Python によるロジスティック写像の描画

```
import numpy as np
from matplotlib import pyplot as plt

class LogisticMap:
    def __init__(self, constant: float, problem_number: int) -> None:
        self.r = constant # constant
        self.x = np.linspace(0, 1, 1000) # length
        self.problem_number = problem_number
        fig = plt.figure(figsize=(12, 6))
        plt.rcParams['font.family'] = 'AppleGothic'
        self.ax1 = fig.add_subplot(1, 2, 1)
        self.ax2 = fig.add_subplot(1, 2, 2)

    def code_problem1(self) -> None:
        ax1_array = []
        # グラフの描画
        for i in self.x:
            num = i
            for _ in range(200):
                num = self.r * num * (1 - num)
            ax1_array.append(num)
        self.ax1.set_xlabel('$x_0$')
        self.ax1.set_ylabel('$x_{200}$')
        self.ax1.set_title(f'r={constant}, と X_0X_200 の関係式')
        self.ax1.plot(self.x, ax1_array, marker='.', linestyle='None')

    def code_problem2(self) -> None:
        ax2_array = []
        x_array = []
        for i in self.x:
            num = i
            for j in range(200):
                num = self.r * num * (1 - num) # formula
                if 150 <= j:
                    ax2_array.append(num)
                    x_array.append(i)
        self.ax2.set_xlabel('$x_0$')
        self.ax2.set_ylabel('$x_{i}:(150 < i < 200)$')
        self.ax2.set_title(f'r={constant}, X_150 ~ X_200 の関係式')
        self.ax2.plot(x_array, ax2_array, marker='.', linestyle='None')

```

```
42      def show_graph(self) -> None:
43          plt.grid()
44          plt.savefig(f"/Users/ryusuke/Documents/git/FutureUniversityHakodate/
                ComplexScienceExercise/result/week02_{self.problem_number}.png")
45
46  r = [1.50, 2.60, 3.20, 3.50, 3.86, 3.90]
47
48  for index, constant in enumerate(r):
49      Log = LogisticMap(constant, index)
50      Log.code_problem1()
51      Log.code_problem2()
52      Log.show_graph()
```

2.4 課題１の考察

r が小さいときには直線を描いているが、r が大きくなるにつれて、無数の点を描画していることが読み取れる。このことから値が大きくなればなるほど、不規則な動きをするということがわかった。
$r = 2.60$ のときに４つの直線が現れることも大きな発見であった。

2.5 課題 2

課題 1 で得られた結果から初期値鋭敏性を説明せよ。

初期値鋭敏性とは、カオスの特性の１つで、たとえ同じ写像や微分方程式であっても初期値に微小な差があれば時間発展とともに指数関数的にその差が大きくなる性質である。
r が少し異なるだけで、グラフが大きく異なるのはこの初期値鋭敏性があるからである。

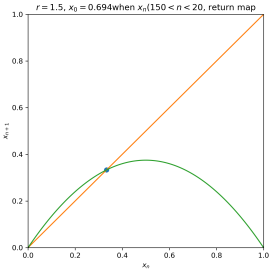
3　複雑系科学演習：レポート課題 3

3.1　課題 1

ロジスティク写像の初期変動の影響がないリターンマップを描くためのプログラムを作成し、個体数変動のリターンマップを表示せよ。
このとき、r は、$r = 1.50$, $r = 2.60$, $r = 3.20$, $r = 3.50$, $r = 3.86$, $r = 3.90$ のとし、初期値 x_0 はランダムに与えなさい。
グラフは、授業資料を参考として、 $x_{n+1} = r(1 - x_n)x_n$, $x_{n+1} = x_n$ も同時に描画し、縦軸と横軸は 0〜1 の範囲で出力すること。

3.2　ロジスティック写像の初期変動の影響がないリターンマップの描画

ロジスティック写像の初期変動の影響がないリターンマッは以下のようになる。



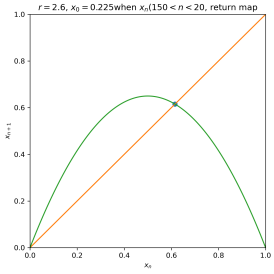
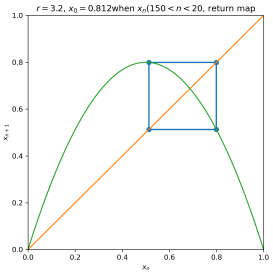


図 19 r=1.50 で初期変動の影響がないリターンマップ　図 20 r=2.60 で初期変動の影響がないリターンマップ



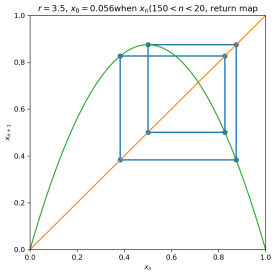
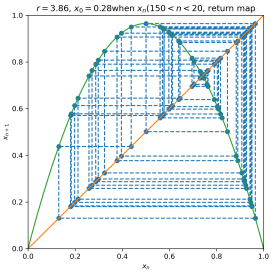


図 21 r=3.20 で初期変動の影響がないリターンマップ　図 22 r=3.50 で初期変動の影響がないリターンマップ



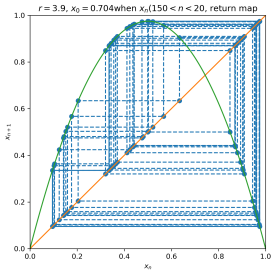


図 23 r=3.86 で初期変動の影響がないリターンマップ　図 24 r=3.90 で初期変動の影響がないリターンマップ

3.3 課題 2

r が 1〜4 のときのロジスティク写像の分岐図を描け。また、分岐図の中で 3 周期の窓が現れている r の範囲を抽出して、グラフを描け。
このとき、両グラフとも r は各自適切な刻み幅を設定し、各 r について初期値 x_0 をランダムに与えること。
プログラムのソースコードは、r が 1〜4 のときの分岐図を出力するものと 3 周期の窓を出力するものとの 2 つを記載すること。

3.4 ロジスティック写像の分岐図の描画

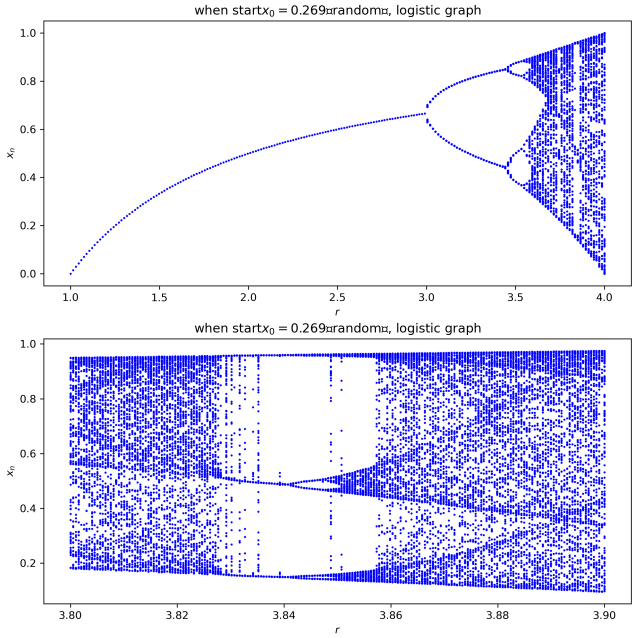


図 25 ロジスティック写像の分岐図の描画

3.5 課題１と課題２のソースコード

Listing 4　Python による課題１と課題２の描画

```
from matplotlib import pyplot as plt
import numpy as np
import random

class Logistic():
    def __init__(self) -> None:
    """create instance"""
        self.r = np.linspace(1, 4, 200, dtype=object)
        self.r2 = np.linspace(3.8, 3.9, 200, dtype=object)
        fig = plt.figure(figsize=(10, 10))
        self.ax1 = fig.add_subplot(2, 1, 1)
        self.ax2 = fig.add_subplot(2, 1, 2)
        self.x = random.uniform(0, 1)

        # Fixed point
        self.u1 = list(map(lambda x: 0, self.r))
        self.u2 = list(map(lambda x: 1 - 1 / x, self.r))
        self.u3 = list(map(lambda x: 0, self.r2))
        self.u4 = list(map(lambda x: 1 - 1 / x, self.r2))

        # multiplier
        self.lambda1 = self.r
        self.lambda2 = list(map(lambda x: 2 - x, self.r))
        self.lambda3 = self.r2
        self.lambda4 = list(map(lambda x: 2 - x, self.r2))

        self.filepath = "/Users/ryusuke/Documents/git/FutureUniversityHakodate/
            ComplexScienceExercise/result/"

    def proprocessing(self, r: float) -> list:
        """Pre-processing to put in the graph"""
        x_array = []
        r_array = []
        num = self.x
        for i in range(150):
            num = r * num * (1 - num)
            if i < 40:
                continue
            x_array.append(num)
            r_array.append(r)
        return x_array, r_array
```

```
41
42      def code_problem1(self):
43          self.ax1.set_title('when start$x_0 = {} ()
                $random, logistic graph'.format(round(self.x, 3)))
44          self.ax1.set_xlabel('$r$')
45          self.ax1.set_ylabel('$x_n$')
46          for i in range(len(self.r)):
47              if -1 <= self.lambda1[i] <= 1:
48                  self.ax1.scatter(self.r[i], self.u1[i], color='b', s=1)
49              elif -1 <= self.lambda2[i] <= 1:
50                  self.ax1.scatter(self.r[i], self.u2[i], color='b', s=1)
51              else:
52                  result = self.proprocessing(self.r[i])
53                  self.ax1.scatter(result[1], result[0], color='b', s=1)
54
55      def code_problem2(self):
56          self.ax2.set_title('when start$x_0 = {} ()
                $random, logistic graph'.format(round(self.x, 3)))
57          self.ax2.set_xlabel('$r$')
58          self.ax2.set_ylabel('$x_n$')
59          for i in range(len(self.r2)):
60              if -1 <= self.lambda3[i] <= 1:
61                  self.ax2.scatter(self.r2[i], self.u3[i], color='b', s=1)
62              elif -1 <= self.lambda4[i] <= 1:
63                  self.ax2.scatter(self.r2[i], self.u4[i], color='b', s=1)
64              else:
65                  result = self.proprocessing(self.r2[i])
66                  self.ax2.scatter(result[1], result[0], color='b', s=1)
67
68      def do_plot(self):
69          self.code_problem1()
70          self.code_problem2()
71          plt.savefig(self.filepath + '課題 week03_2', dpi=300)
72
73  class Report_3():
74      def __init__(self, r: float, s: str) -> None:
75          self.r = r
76          self.s = s
77          self.x = random.uniform(0, 1)
78          self.xn = np.linspace(0, 1, 1000)
79          self.filepath = "/Users/ryusuke/Documents/git/FutureUniversityHakodate/
                ComplexScienceExercise/result/"
80
81      def logistic(self, x: float, cnt: int) -> list:
82          num = x
83          num_array = [x]
```

```
84          for _ in range(cnt):
85              num = self.r * num * (1 - num)
86              num_array.append(num)
87          return num_array
88
89      def pre_processing(self, cnt: int) -> float:
90          """Pretreatment to eliminate the effects of initial fluctuations"""
91          num = self.x
92          for _ in range(cnt):
93              num = self.r * num * (1 - num)
94          return num
95
96      def plot_return_map(self):
97          """Drawing a return map that is not affected by initial fluctuations"""
98          plt.figure(figsize=(6, 6))
99
100         x = self.pre_processing(150)
101         n = self.logistic(x, 50)
102         xn_array = []
103         for i in self.xn:
104             xn_array.append(self.r * (1 - i) * i)
105
106         # net map
107         spiper_plot_x = []
108         spiper_plot_y = []
109         for i in range(1, len(n)):
110             spiper_plot_x.append(n[i - 1])
111             spiper_plot_x.append(n[i])
112             spiper_plot_y.append(n[i])
113             spiper_plot_y.append(n[i])
114
115         plt.plot(spiper_plot_x, spiper_plot_y, marker='o', linestyle='dashed')
116         plt.plot(self.xn, self.xn)
117         plt.plot(self.xn, xn_array)
118         plt.title("$r = $" + str(self.r) + ", $x_0 = $" + str(round(self.x, 3)) + '
                when $x_n (150 < n < 20 $, return map')
119         plt.xlim(0, 1)
120         plt.ylim(0, 1)
121         plt.xlabel("$x_n$")
122         plt.ylabel("$x_{n+1}$")
123         print('hoge', self.filepath + self.s)
124         plt.savefig(self.filepath + self.s, dpi=300)
125
126 r = [1.50, 2.60, 3.20, 3.50, 3.86, 3.90]
127
128 # assignment1
```

```
129 for index, num in enumerate(r):
130     assignment_1 = Logistic(num, "課題 week03_1_{}".format(index + 1))
131     assignment_1.plot_return_map()
132 
133 # assignment2
134 assignment_2 = Logistic()
135 assignment_2.do_plot()
```

3.6　課題１と課題２に関する考察

カオスの持つ性質の初期値鋭敏性によって、 初期値によりグラフの形が大きく変化する。しかし、試行回数を増やすことによって、最終的には固定点に収束することがわかる。よって、試行回数を増やすのが特徴を見る上では重要であると考えた。
また、$1 \leq r \leq 3$ の範囲では固定点に収束していることがわかり、第２回の課題で $r = 1.50$ や $r = 2.60$ の時に値が一つしかなかった根拠が分岐図からわかる。同様に $r \leq 3.28, 3.86 \leq r$ のときに収束先が複数あることから、カオス状態になっていることがわかる。また、収束先が３つあることから３周期解であることが読み取れた。

3.7 課題 3

課題 1 と課題 2 は、各 r ごとに初期値 x_0 をランダムに与えているにもかかわらず、r が 1〜3.5 くらいまでは何度プログラムを実行しても同じようなグラフを描くことができる。
一方、r が 3.5 よりも大きくなっていくと、プログラムを実行するたびにグラフを一見するだけではわからないような違いが生じる。
この理由を前回の課題と初期値鋭敏性という言葉とを用いて説明せよ。

分岐図から見てもわかるよう r に比例して分岐の数が大きくなっていく。$3.5 \leq r$ の場合だとカオス状態となっている。したがってそこではカオス特有の初期値鋭敏性が働く。ランダムに初期値をとった場合、初期値鋭敏性によって収束する場所が変わってくるので $3.5 \leq r$ の場合は出力するごとに少しずつ変化するのではないかと考えた。

4　複雑系科学演習：レポート課題 4

4.1　課題 1

　　リアプノフ指数の r 依存性を示したグラフを描け。
但し、初期値をランダムに与え、グラフの横軸は 1〜4、縦軸は -3〜1 までの範囲にすること。
r の刻み幅は、各自適切な値を設定すること。

4.2　課題 2

　3 周期の窓の領域でのリアプノフ指数の r 依存性を示したグラフを描け。
但し、初期値をランダムに与え、グラフの縦軸は -1〜0.4 までの範囲にすること。
r の範囲および刻み幅は、各自適切な値を設定すること。

4.3　課題１と課題２のリアプノフ指数の依存性を示したグラフ描画

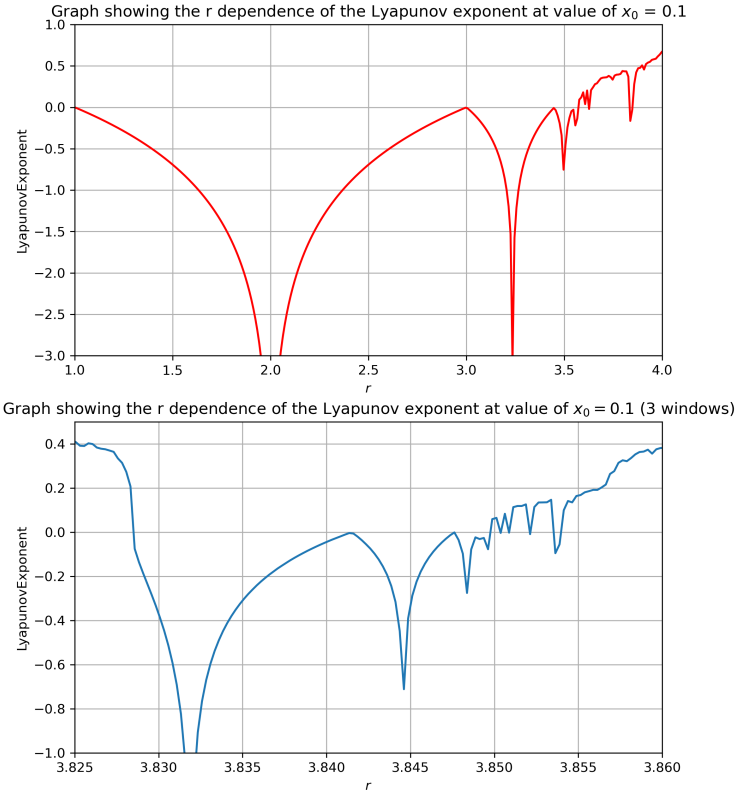


図 26 リアプノフ指数の依存性を示したグラフ描画

4.4 リアプノフ指数の依存性を示したグラフのソースコード

Listing 5 Python によるリアプノフ指数の依存性を示したグラフの描画

```
from matplotlib import pyplot as plt
import numpy as np
from math import log

class LyapunovExponent():

    def __init__(self) -> None:
        self.r = np.linspace(1, 5, 400, dtype=np.object) # constant1
        self.r2 = np.linspace(3.8, 3.9, 400, dtype=np.object) # constant2
        fig = plt.figure(figsize=(8, 10))
        self.ax1 = fig.add_subplot(2, 1, 1)
        self.ax2 = fig.add_subplot(2, 1, 2)
```

```
13          self.ax1.grid(True)
14          self.ax2.grid(True)
15          self.x = 0.1
16
17      def f(self, r: float, x: float) -> float:
18          "A function that returns logistic regression (initial value of x is 0.1)"
19          return r * x * (1 - x)
20
21      def f_prime(self, r: float, x: float) -> float:
22          "A function that returns the derivative of logistic regression (initial value of
                x is 0.1)"
23          return r * (1 - 2 * x)
24
25      def calc_lambda(self, r: float) -> float:
26          "Function to calculate the multiplier λ ()"
27          lambda_sum = 0
28          x_array = []
29          cnt = 10000
30          num = self.x
31
32          # preprocessing
33          for i in range(300):
34              num = self.f(r, num)
35
36          for i in range(cnt):
37              x_array.append(num)
38              num = self.f(r, num)
39
40          for i in x_array:
41              if self.f_prime(r, i) > 0: lambda_sum += log(self.f_prime(r, i))
42              else: lambda_sum += log(-1 * self.f_prime(r, i))
43
44          return lambda_sum / cnt
45
46      def problem1(self) -> None:
47          "Drawing a graph showing the r dependence of the Lyapunov exponent"
48          lambda_array = []
49          for i in self.r:
50              lambda_array.append(self.calc_lambda(i))
51          self.ax1.plot(self.r, lambda_array, color='red')
52          self.ax1.set_title("Graph showing the r dependence of the Lyapunov exponent at
                value of $x_0$ = 0.1")
53          self.ax1.set_xlim(1, 4)
54          self.ax1.set_xlabel('$r$')
55          self.ax1.set_ylim(-3, 1)
56          self.ax1.set_ylabel('LyapunovExponent')
```

```
57
58      def problem2(self) -> None:
59          "Drawing a graph showing the r dependence of the Lyapunov exponent in the three-
                cycle window area"
60          lambda_array = []
61          for i in self.r2:
62              lambda_array.append(self.calc_lambda(i))
63          self.ax2.plot(self.r2, lambda_array)
64          self.ax2.set_title(
65              "Graph showing the r dependence of the Lyapunov exponent at value of $x_0 =
                    0.1$ (3 windows)")
66          self.ax2.set_xlim(3.825, 3.86)
67          self.ax2.set_xlabel('$r$')
68          self.ax2.set_ylim(-1, 0.5)
69          self.ax2.set_ylabel('LyapunovExponent')
70
71      def save_fig(self):
72          file_path = '/Users/ryusuke/Documents/git/FutureUniversityHakodate/
                ComplexScienceExercise/result/week04_1019163'
73          plt.savefig(file_path, dpi=300)
74          plt.show()
75
76      def plot(self):
77          self.problem1()
78          self.problem2()
79          self.save_fig()
80
81  Lyapunov = LyapunovExponent()
82  Lyapunov.plot()
```

4.5　課題１と課題２に関する考察

　２つのグラフを比べた際にグラフの概形が似ていることに気づいた。これはリアプノフ指数に依存しているためではないかと考えた。
3 周期の窓のグラフに着目すると $3.83 \leq r \leq 3.84$ の領域ではリアプノフ指数が負になっており、カオス状態ではないことがわかる。

4.6 課題 3

ロジスティック写像についてまとめ、これまでに出題された全て (4 回分) の結果について考察せよ。
分量は $A4$ 用紙 1〜4 枚程度を目安としてください。

ロジスティック写像とは離散型カオスを観測できる写像である。時系列データを観測したことにより、生物の個体数が収束する時間は最大増加率に比例することがわかった。しかし、収束しないものも中には存在し、最大増加率に依存して時系列データが大きく変化することが確認できた。
また、初期値をわずかでも変更すると、一定時間が経過したら大きな変化が生じることが確認できた。初期値のパラメータが小さい時には一定時間が経過することである値に収束することが確認できたが、パラメータが大きくなると、大きなずれが生じることがわかった。これがカオスの性質の一つである初期値鋭敏性である。
初期値鋭敏性は日常でも起きている場面があり、それが振り子である。振り子は落下位置、落下速度、落下角度のどれか一つでもわずかに変更するだけで全く違う挙動を描く。これは初期値のわずかなずれが時間の経過とともに大きな変化となっていく初期値鋭敏性の性質が起きているためである。
このようにカオスの性質は日常でも存在していることがわかった。
次に、ロジスティック写像のリターンマップを初期値をランダムに撮った際にどのような軌道を描くのかを調べた。すると、大きな変化ではないが、初期値鋭敏性によりリターンマップの形がわずかにずれてくることがわかった。それを防ぐために初期値鋭敏性の影響を受けないようある程度収束した状態のグラフを描画した。
リターンマップについては、r の値が大きくなるにつれて、収束するために多くの反復試行を繰り返す必要があることが確認できた。
また、この写像の分岐図を描画し、どのような現象が起っているのかを観測した。 分岐図とはパラメータによって収束する場所がどうなるかをグラフ化したものである。このグラフから r の値が大きくなるにつれ、より多くの場所に収束することがわかる。この図を見ることで収束場所が無数にあるカオス状態とそうでないかをはっきりと区別することができる。
この写像はパラメータが増えるごとに収束する場所が倍に増えていく周期倍分岐という特徴を持っている。それを確認するための指標となる数値のことをリアプノフ指数と言い、この値が正である時にカオスになる。実際に r との関係をグラフにし、可視化することでどの部分がカオス状態なのかが確認できた。グラフを見てみるとリアプノフ指数が正となっているのは $3.56 \leq r$ 程度のときで、その領域がカオスになっているのはそれまでの課題から確認できる。また、3 周期の窓の部分はリアプノフ指数が負になっているので周期解が得られることがわかる。

5　複雑系科学演習：レポート課題 5

5.1　課題 1

Mandelbrot 集合を描け。

5.2　Mandelbrot 集合の描画

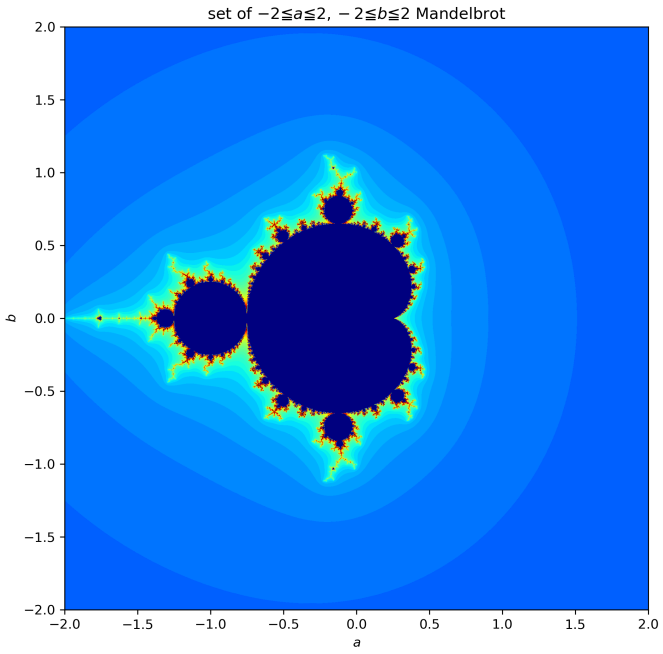


図 27 Mandelbrot 集合の描画

5.3 Mandelbrot 集合のソースコード

Listing 6 Python によるマンデルブロ集合の描画

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.colors import Normalize
from numba import jit

@jit
def mandelbrot(a, b, n_max):複素数を用いてマンデルブロ集合の座標を求める関数
    ""
    a_num, b_num = np.meshgrid(a, b)
    n_grid = len(a_num.ravel()) # Total number of combinations
    z = np.zeros(n_grid) # Empty array for data storage of Mandelbrot set

    for i in range(n_grid):
        c = complex(a_num.ravel()[i], b_num.ravel()[i]) # c = a + bi
        n = 0
        z0 = complex(0, 0)
        while np.abs(z0) < np.inf and not n == n_max:
            z0 = z0 ** 2 + c # Calculate recurrence formula
            n += 1

        if n == n_max:
            z[i] = 0
        else:
            z[i] = n
    z = np.reshape(z, a_num.shape) # Convert to a two-dimensional array
    z = z[::-1] # Since it is turned upside down with imshow (), it is turned upside
        down.
    return z

a = np.linspace(-2, 2, 2000)
b = np.linspace(-2, 2, 2000)
z = mandelbrot(a, b, 100)
file_path = '/Users/ryusuke/Documents/git/FutureUniversityHakodate/
    ComplexScienceExercise/result/week05_1_1019163'
plt.figure(figsize=(8, 8))
plt.xlabel('$a$')
plt.ylabel('$b$')
plt.title("set of $-2 \leqq a \leqq 2, -2 \leqq b \leqq 2$ Mandelbrot")
plt.imshow(z, cmap='jet', norm=Normalize(vmin=0, vmax=50), extent=[-2, 2, -2, 2])
plt.savefig(file_path, dpi=300)
plt.show()
```

5.4 課題 2

Julia 集合を描け。

5.5 Julia 集合の描画

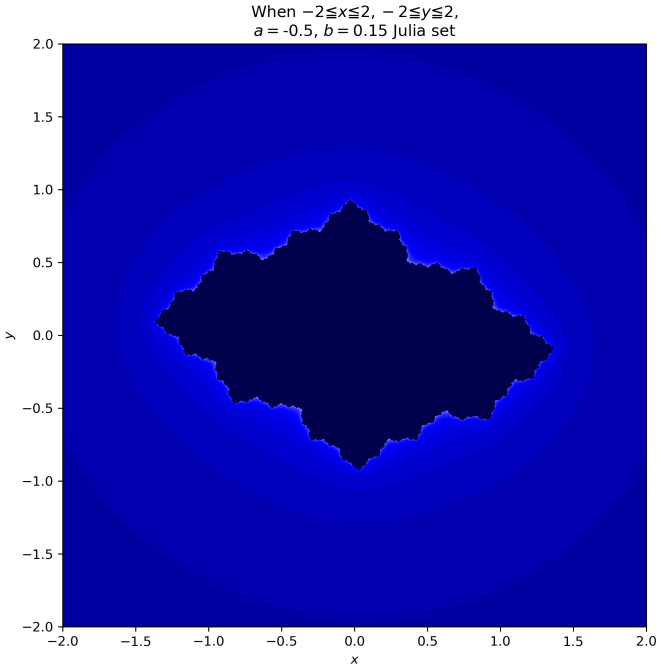


図 28 Julia 集合の描画

5.6 Julia 集合のソースコード

Listing 7 Python によるジュリア集合の描画

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.colors import Normalize
from numba import jit

@jit
def julia(x, y, n_max, a, b):
    "A function that finds the coordinates of the Julia set using complex numbers"
    a_num, b_num = np.meshgrid(x, y)
    n_grid = len(a_num.ravel()) # Total number of combinations
    z = np.zeros(n_grid) # Empty array for data storage of Mandelbrot set

    for i in range(n_grid):
        c = complex(a, b) # c = a + bi

        n = 0
        z0 = complex(a_num.ravel()[i], b_num.ravel()[i])

        while np.abs(z0) < 1e20 and not n == n_max:
            z0 = z0 ** 2 + c # Calculate recurrence formula
            n += 1

        if n == n_max:
            z[i] = 0
        else:
            z[i] = n

    z = np.reshape(z, a_num.shape) # Convert to a two-dimensional array
    z = z[::-1] # Since it is turned upside down with imshow (), it is turned upside
        down.
    return z

x = np.linspace(-2, 2, 2000)
y = np.linspace(-2, 2, 2000)
n_max = 50

a, b = -0.5, 0.15
z = julia(x, y, n_max, a, b)
file_path = '/Users/ryusuke/Documents/git/FutureUniversityHakodate/
    ComplexScienceExercise/result/week05_2_1019163'
plt.figure(figsize=(8, 8))
```

```
40 plt.xlabel('$x$')
41 plt.ylabel('$y$')
42 plt.title("When $-2 \leqq x \leqq 2, -2 \leqq y \leqq 2$,\n" +\
43     "$a = ${0}, $b = ${1} Julia set".format(round(a, 3), round(b, 3)))
44 plt.imshow(z, cmap='seismic', norm=Normalize(vmin=0, vmax=n_max), extent=[-2, 2, -2,
       2])
45 plt.savefig(file_path, dpi=300)
46 plt.show()
```

5.7 課題 3

Koch 曲線の描け。

5.8 コッホ曲線の描画

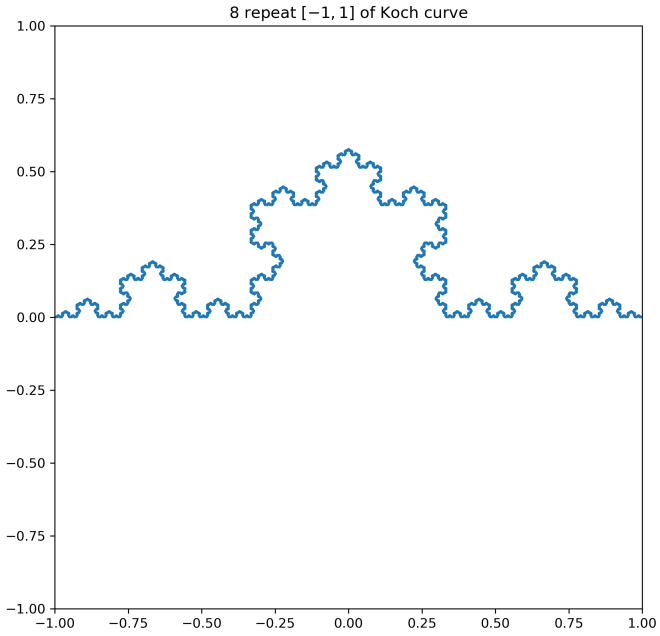


図 29 コッホ曲線の描画

5.9 コッホ曲線のソースコード

Listing 8　Python によるコッホ曲線の描画

```
import math
from matplotlib import pyplot as plt

def koch(n: int, p1: list, p2: list) -> None:
    "A function that finds the coordinates of a Koch curve using a recursive function"

    if n == 0: return

    sx = 2 * p1[0] / 3 + p2[0] / 3
    sy = 2 * p1[1] / 3 + p2[1] / 3
    tx = p1[0] / 3 + 2 * p2[0] / 3
    ty = p1[1] / 3 + 2 * p2[1] / 3
    ux = (tx - sx) * math.cos(math.radians(60)) - (ty - sy) * math.sin(math.radians
        (60)) + sx
    uy = (tx - sx) * math.sin(math.radians(60)) + (ty - sy) * math.cos(math.radians
        (60)) + sy
    koch(n - 1, p1, [sx, sy])
    koch_array_x.append(sx)
    koch_array_y.append(sy)
    koch(n - 1, [sx, sy], [ux, uy])
    koch_array_x.append(ux)
    koch_array_y.append(uy)
    koch(n - 1, [ux, uy], [tx, ty])
    koch_array_x.append(tx)
    koch_array_y.append(ty)
    koch(n - 1, [tx, ty], p2)

n = 8
p1, p2 = [-1, 0], [1, 0]
koch_array_x, koch_array_y = [p1[0]], [p1[1]]
koch(n, p1, p2)
koch_array_x.append(p2[0])
koch_array_y.append(p2[1])
file_path = '/Users/ryusuke/Documents/git/FutureUniversityHakodate/
    ComplexScienceExercise/result/week05_3_1019163'
plt.figure(figsize=(8, 8))
plt.title("{} repeat $[-1, 1]$ of Koch curve".format(n))
plt.xlim(-1, 1)
plt.ylim(-1, 1)
plt.plot(koch_array_x, koch_array_y)
plt.savefig(file_path, dpi=300)
```

```
39 plt.show()
```

5.10　レポート課題５に関する考察

マンデルブロ集合、ジュリア集合、コッホ曲線は名称自体は知っていたが、実際に Python を用いて描画したのは初めてだったためとても苦労した。コードの中に複素数を取り入れた部分に関しては、調べながら実装してなんとかできた。
ジュリア集合は初期値の a, b を少し変更するだけで全く違う図形が現れたことは大きな発見で合った。これも初期値鋭敏性によるカオスの性質であるものと考えることができる。
コッホ曲線は自己相似図形でありフラクタル構造を持つことで有名である。このフラクタル図形は日常でもリアス式海岸などが有名である。海岸の長さを図ろうとすると、縮小すればするほど細かいくぼみなどがあり長さが無限大になることが知られている。
マンデルブロ集合に関しても、実際に描画するのは初めての経験だったため良い経験となった。漸化式を定義する部分が難しく、調べながら行った。

6　最後に

5 回分の講義についての感想・意見等があれば、書いてください。(採点対象ではありません。)

第５回のマンデルブロ集合、ジュリア集合、コッホ曲線は名称自体は知っていたが、実際に Python を用いて描画したのは初めてだったためとても苦労した。コードの中に複素数を取り入れた部分に関しては、調べながら実装してなんとかできた。
コッホ曲線などの自己相似図形はフラクタル図形とも呼ばれ、前期の講義のカオスフラクタルでも自習で勉強していた部分があったのである程度は書くことができた。フラクタル図形に関しては、シェルビンスキーの三角形やバーンズレーのシダなどもある。このような図形についても今後の複雑系科学演習で扱うことができれば良い学習になるのではと思った。
また、この講義は課題が公開されてからほぼ半日で図形を描画しなければならないのがかなり厳しいと感じた。自分自身は１年の頃から毎日 Python を触っていたので苦労することなく描画することが出来たが、周囲の友人などにはわからないから教えて欲しいなどとサポートを頼まれることもよくあった。せめて１週間など期間を延長できれば、学生にとってより良い学習になるのではないかと思った。